文章编号:1003－6199(2017)02－0136－05

工厂设计模式的研究与应用

葛　萌†,欧阳宏基
(咸阳师范学院 计算机学院,陕西 咸阳　712000)

摘　要:为了提高传统JDBC框架的复用性,分析了工厂设计模式的三种具体形式:简单工厂、工厂方法和抽象工厂。阐述了三者之间的优缺点,从进化和退化两个方面分析了三者之间的转换关系。将工厂设计模式与JDBC相结合,设计了一个数据持久层模型,给出了该模型的设计思想与若干核心代码。通过相关分析与测试表明:将工厂设计模式应用到持久层的设计中能够减少代码的冗余度、提高复用性和扩展性。

关键词:简单工厂;工厂方法;抽象工厂;JDBC
中图分类号:TP311.1　　文献标识码:A
DOI:10.16339/j.cnki.jsjsyzdh.2017.02.029

Research and Implementation of Factory Design Pattern

GE Meng,OUYANG Hong-ji
(Computer College,Xianyang Normal University,Xianyang,Shaanxi 712000,China)

Abstract:In order to improve the reusability of the traditional JDBC framework,this paper analyzes three concrete forms of the factory design pattern,which are simple factory,factory method and abstract factory. expounds the advantages and disadvantages of the three,the transformation relationship between the three is analyzed from two aspects of evolution and degradation.designs a data persistence layer model with combining factory design pattern and JDBC,gives the design idea and some core codes of the model.Through the correlation analysis and test,it is indicated that the factory design pattern can be applied to the design of persistent layer,which can reduce the redundancy of the code,improve the reusability and expansibility.

Key words:simple factory;factory method;abstract factory;JDBC

0　引　言

工厂设计模式属于创建型模式中使用最为频繁的一种,它的主要思想是将对象的创建封装到一种称为“工厂”的类中,从调用方角度来看,需要“产品”时,不需要亲自创建出来,通过调用工厂对象的方法就可以得到对象。因此,合理的使用工厂设计模式能够将对象的创建和使用相分离,从而减少类之间的耦合度,提高复用性。本文首先介绍了工厂设计模式中的三种具体形式:简单工厂模式、工厂方法模式和抽象工厂模式,详细描述了每种形式的组成以及各角色在模式中承担的功能,从进化和退化两个方面分析了它们三者之间的转换关系。最后,以JDBC作为Java EE应用持久层解决方案的背景下,将工厂模式的三种具体形式应用到持久层的设计过程中,提出了一个数据持久层模型,对该模型的设计过程进行了分析,通过相关测试证明了它的有效性。

1　工厂设计模式分析

1.1　简单工厂模式

简单工厂模式包含三个角色[1]:抽象产品、具

收稿日期:2016－07－29
基金项目:咸阳师范学院专项科研计划项目(14XSYK038)
作者简介:葛萌(1980—),女,陕西咸阳人,讲师,硕士,研究方向:软件工程。
†通讯联系人,E-mail:oyhj_nicholas@163.com

体产品和工厂。抽象产品角色是工厂所创建的所有对象的共同父类,描述了所有产品的公共接口。具体产品是该模式的创建目标,所有创建的对象都充当这个角色的某个具体类的实例。工厂角色对外提供一个静态的工厂方法用来创建所有的具体产品,通过参数动态决定所创建产品的类型,方法内部针对参数形成判断逻辑。当产品类型发生变化时会导致判断逻辑的变化,所以简单工厂模式不满足“开闭原则”。

1.2　工厂方法模式

工厂方法模式[2]一共包含 4 个角色:抽象产品、具体产品、抽象工厂和具体工厂。抽象产品是产品对象的共同父类或接口。具体产品由某种类型的具体工厂所创建。抽象工厂用来声明工厂方法并返回产品。具体工厂用来创建一个具体的产品类对象,其中包含了与应用程序密切相关的逻辑。具体工厂与具体产品一一对应。相对于简单工厂,工厂方法在工厂这一侧进行了抽象,将具体产品的创建延迟到工厂子类中进行。如果系统中引入了新的产品,那么只需要创建新的产品和新的工厂即可,系统中原来的产品和工厂类不需要修改,所以工厂方法很好的满足了“开闭原则”。

1.3　抽象工厂模式

工厂方法模式中只能生产一种类型的产品,当产品种类多于一种时,工厂方法模式就不满足“开闭原则”,此时只能使用抽象工厂模式。理解抽象工厂模式首先要明确两个概念:产品等级结构和产品族[3-4]。前者表示产品一侧的泛化关系,后者表示同一个工厂所生产的、位于不同等级结构中的一组不同种类的产品。抽象工厂定义一组生成抽象产品的方法,每个方法对应一个产品等级结构。具体工厂生产一组具体产品形成一个产品族,每一个产品都位于某个产品等级结构中。抽象产品用于定义产品的抽象业务。具体工厂生产具体产品对象。

1.4 三者的优缺点及转换

工厂设计模式的核心在于将对象的创建和对象本身业务处理相分离,降低系统的耦合度,使两者的修改变得简单。简单工厂模式将所有产品的创建过程封装到工厂类的静态方法中,通过传入正确的参数即可获得所需对象。但是工厂类的任务相对繁重,尤其是在产品类过多的情况下,工厂类会有繁琐的判断逻辑;而且增加新产品的同时需要修改判断逻辑。

工厂方法模式在工厂一侧引入了泛化关系,它的实现依赖于工厂角色与产品角色的多态性。把原来集中创建产品对象的方式改为分散式创建,每一个具体工厂创建每一种具体产品。如果有新产品的加入,只需增加具体产品类和对应的具体工厂类即可,原来的代码无需更改。

但是工厂方法模式只能创建类型单一的产品,当产品类型增多时,系统中类的个数成对增加,提高了系统的复杂度和编译开销。

抽象工厂模式解决了工厂方法模式所创建产品种类单一的问题,它提供一个创建一系列相关或相互依赖对象的接口,而无须指定它们具体的类[5-6]。产品等级结构决定了产品种类的个数,产品族决定了具体工厂的个数。从产品族的角度而言,增加新的具体工厂时无须修改原有代码,满足开闭原则;从产品等级结构的角度而言,增加新的产品类型时需要修改其中的抽象工厂角色代码,同时还要修改各个具体的工厂类。所以,抽象工厂模式对于开闭原则具有半倾斜性[7]。它们三者的优缺点及转换关系见表 1。

表 1　工厂设计模式的优缺点及转换关系表

模式名称	优点	缺点	进化	退化
简单工厂模式	集中创建所有产品,通过简单参数与具体产品对应	工厂角色职责繁重,不满足开闭原则	对工厂进行抽象,即可进化为工厂方法模式	无
工厂方法模式	工厂角色进行了抽象,增加新产品时满足开闭原则。	每个具体工厂只能生产一种产品	具有两个及其以上数量的产品种类,即可转换为抽象工厂模式	工厂等级结构中只有一个具体工厂类,即退化为简单工厂
抽象工厂模式	一个产品族中的多个对象被设计成一起工作时,能够确保调用端只使用同一个产品族中的对象。	高度抽象,难于理解。扩展产品等级结构时不满足开闭原则。	无	产品等级结构的个数为 1 时,退化为工厂方法模式。

2 工厂设计模式在 Java EE 持久层的应用

Java EE 的持久层用来封装数据持久化逻辑并为业务层提供访问数据源的接口，提供诸如数据源连接、查询、存储过程、数据格式修正和错误处理等功能[8]。其目的是为了解耦合业务处理和数据存取，为企业应用形成一个高效、稳定的数据访问环境。由于关系型数据库在数据存储方面仍然占据主导地位，所以围绕 SQL 产生出了很多数据持久层解决方案：包括 JDBC、全自动化的 ORM(例如 Hibernate)、半自动化的 ORM(例如 MyBatis)以及 JDO 等。其中 JDBC 是最原生态的 SQL 解决方案，具有执行效率最高、易于掌握等特点，但也有复用率低、不易扩展等缺点。本节主要讨论如何将工厂设计模式应用到 JDBC 中并设计一个数据持久层模型。

2.1 抽象工厂模式的应用

结合应用背景，分析出产品等级结构和产品族是应用抽象工厂模式的关键。为了隔离业务逻辑与持久化逻辑，Java EE 规范推荐采用 DAO 模式。通常的做法是在 DAO 接口中定义相关的持久化方法，DAO 实现类中应用某种具体的持久化技术来完成持久化方法[9]。由于一个系统中存在多个不同的实体对象，它们所对应的 DAO 可以看作产品等级结构；由于不同数据库具有 SQL“方言”，在执行相同的持久化逻辑时 SQL 语句会有所差别，因此在某个特定数据库下的各种 DAO 的实现类可以看作一个产品族。业务层要对实体对象进行持久化操作必须通过工厂获取对应的 DAO 对象。以 MySQL 数据库为例，部分角色的代码如下：

```
public interfaceDAOFactory
{    //定义系统中所有实体对象的 DAO
UserDAO  createUserDAO();
DepartmentDAO createDepartmentDAO();
……………………
}
```

每个具体的数据库对应一个具体工厂，代码如下：

```
public classMySQLDAOFactory implements DAOFactory
{
public UserDAO createUserDAO() {
return new MySQLUserDAOImp();
}
publicDepartmentDAO createDepartmentDAO() {
return new MySQLDepartmentDAOImp();
}
…………………………………………
}
```

2.2 工厂方法模式的应用

JDBC 的操作一般包括 4 个步骤[10]：(1)加载驱动；(2)获取 Connection；(3)创建相关 Statement 对象并执行 SQL 语句；(4)释放资源。为了减少代码的冗余度，定义抽象类 JDBCUtil 用来执行步骤(1)、(2)和(4)，将该类对象看作工厂模式中的唯一抽象产品，定义 JDBCUtilFactory 当作工厂方法模式中的抽象工厂，每种具体数据库对应一个 JDBCUtil 和 JDBCUtilFactory 的实现类，分别当作具体产品和具体工厂。根据上述分析，具体产品角色代码如下：

```
public classMySQLJDBCUtil extends JDBCUtil
{   static{
Class.forName(" com.mysql.jdbc.Driver");

…………….}
public ConnectiongetConnection() throws SQLException{
return DriverManager.getConnection(" jdbc:mysql://localhost: 3306/dbName "," root "," root");
}
```

具体工厂角色代码如下：

```
public classMySQLJDBCUtilFactory implements JDBCUtilFactory
{
publicJDBCUtil createJDBCUtil() {
return new MySQLJDBCUtil();
}
}
```

2.3 简单工厂模式的应用

通过对 2.3 节的代码分析可以看出：不同数据库所对应的具体产品和具体工厂的代码结构相同，不同之处在于 JDBC 驱动的名称和创建 Connection 对象时传入的 URL 参数。为了进一步减少冗余度，将这些参数定义到配置文件中，在

程序运行时通过读取配置文件动态传入。这样的话,对于工厂方法模式而言,产品和工厂就不存在抽象层,从而退化成为简单工厂模式。

首先,定义读取配置文件的类-JDBCConfigReader,其中关联一个 Properties 对象,该对象用于读取 properties 类型的配置文件。properties 类型的配置文件具有易于理解、读写简单等特点,以<K,V>键值对形式存放数据。由于配置文件中的内容只需读取一次,放入内存供其它对象使用,所以 JDBCConfigReader 采用单例模式封装。

然后,定义简单工厂模式中的产品和工厂类。产品类的代码如下:

```
public classJDBCUtil
{static{
Class. forName ( JDBCConfigReader. getInstance ( ). getProperties ( ). getProperty (" DriverClass") ;
}
public ConnectiongetConnection ( ) throws SQLException{
String url = JDBCConfigReader. getInstance ().getProperties().getProperty(" DBURL") ;
String userName=JDBCConfigReader.getInstance().getProperties().getProperty(" DBUserName") ;
String password=JDBCConfigReader.getInstance().getProperties().getProperty(" DBPassword") ;
  return DriverManager.getConnection(url, userName,password) ;
} }
```

工厂类的代码如下:

```
publicclass JDBCUtilFactory
{
public static JDBCUtil createJDBCUtil(){
      return newJDBCUtil() ; }
}
```

与 2.3 节的代码对比可以看出:将不同数据库的相关 JDBC 参数存储到配置文件后,工厂方法模式退化成了简单工厂模式,产品和工厂两个角色都变成了一个对象,不但减少了产品类和具体工厂类的个数,而且工厂类在创建产品对象时也避免了逻辑判断。

2.4　业务层对持久层的调用

假定当前系统中的一个实体对象是 User,它对应的 DAO 实现类的代码如下:

```
public classMySQLUserDAOImp implements UserDAO
{
    //通过 JDBCUtil 工厂得到 JDBCUtil 产品
privateJDBCUtil jdbcUtil = new JDBCUtilFactory().createJDBCUtil() ;
//相关实体类的持久化方法
publicboolean addUser(User user)
{
Connection con=jdbcUtil.getConnection() ;
………………………
}
}
```

将当前实际使用的数据库所对应的 DAO 工厂类信息写到配置文件中,将抽象工厂模式中的具体工厂当作简单工厂模式中的具体产品,通过反射机制创建出具体的 DAO 工厂,如下代码所示:

```
public classDAOFactory
{
        public staticDAOFactory getDAOFactory()
    {  DAOFactory factory=null;
String DAOFactoryName = JDBCConfigReader. getInstance ( ). getProperties ( ). getProperty (" DAOFactory") ; factory = ( DAOFactory ) Class. forName (DAOFactoryName).newInstance() ; }
      return factory;
    } }
```

当业务层要获取相关实体的 DAO 对象时,执行下面代码:

```
DAOFactory factory = DAOFactoryConfig. getDAOFactory() ;
UserDAO userDAO = factory. getUserDAO () ;
```

通过上述分析可以看出:业务层对于持久层方法的调用,首先通过简单工厂读取配置文件得到抽象工厂模式的具体 DAO 工厂,然后将具体 DAO 工厂生产的 DAO 产品赋值给抽象 DAO,通过抽象 DAO 调用相关实体的持久化方法。这时与业务层进行通信的只是抽象 DAO 工厂和抽象 DAO 产品。业务层不需要知道当前 DAO 对象由哪个具体的工厂创建。因此,不论使用哪种数据库,对

业务层的调用来说没有任何影响，满足持久层支持多数据库的要求。综合上述，本文设计的持久层模型如图 1 所示。

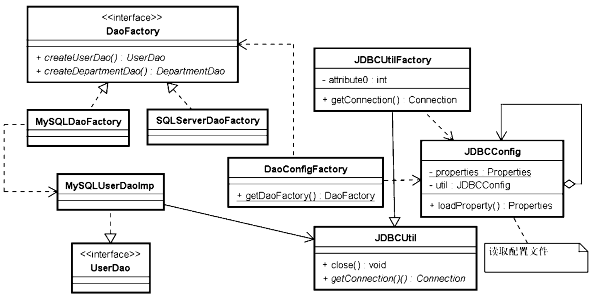

图 1　基于工厂设计模式与 JDBC 的持久层类图

2.5　工厂设计模式应用评价

将本文设计的数据持久层模型与传统 JDBC 进行比较，观测点为执行效率与复用率。其中不包括 DAO，因为 DAO 的代码与具体业务相关，测试工具为 JUnit 和 JDK 的 Executor 并发框架。在单机环境下采用单线程和多线程（并发量为 50）两种形式，执行时间为多次执行的平均值，测试结果见表 2 和表 3。可以看出工厂设计模式的应用提高了代码的复用率，尤其是简单工厂模式＋反射读取配置文件来创建对象的方式，完全可以复用。在执行效率方面，本文设计的持久层模型与传统 JDBC 的执行开销差别很小。在多线程情况下，本文模型效率略有提高。这表明工厂设计模式在提高复用率的情况下，虽然增了的类与对象的调用开销，但对性能的影响可以忽略，因此本模型是有效、可靠的。

表 2　持久层代码复用情况对比表

名称	代码总行数	可复用代码总行数	可复用率
传统 JDBC	65	20	30％
本文持久层模型	133	58	43％

表 3　持久层代码执行时间对比表

名称	单线程方式（单位：秒）	多线程方式（单位：秒）
传统 JDBC	0.345S	0.165S
本文持久层模型	0.355S	0.115S

3　结　论

本文对工厂设计模式进行了研究，分析了他们的优缺点和转换关系。将工厂设计模式与 JDBC 相结合，提出了一种数据持久化模型。通过实际测试表明工厂设计模式能够很好的将对象的创建和使用相分离，向调用方屏蔽对象的创建过程，在不增加过多额外开销的情况下，提高了代码的复用率、扩展性和维护性。为开发人员在设计过程中合理使用工厂设计模式提供了一定的参考。

参考文献

[1]　薛桂香，任女尔，闫世峰，等. 基于简单工厂模式的 SSH＋ExtJs 架构泛型化研究[J].河北工业大学学报，2015，44(3)：65－69.

[2]　华铨平，庞倩超，谢颖.抽象工厂设计模式在 3 层结构开发中的应用[J].大庆石油学院学报，2009，33(3)：112－115.

[3]　郭永平，刘淑娟.工厂方法模式在软件开发中的应用—以监控数据接收服务程序为例[J].宝鸡文理学院学报：自然科学版，2015，35(4)：58－62.

[4]　欧建斌.工厂设计的模式研究[J].微型电脑应用，2010，26(12)：15－17.

[5]　欧阳宏基，葛萌，陈伟.一种改进的建造者设计模式[J].咸阳师范学院学报，2014，29(6)：43－46.

[6]　程裕强.抽象工厂模式探讨[J].玉林师范学院学报，2014，35(2)：82－86.

[7]　刘伟.设计模式[M].北京：清华大学出版社，2011：92－103.

[8]　尚鲜连.设计模式在数据持久层设计中的应用[J].重庆科技学院学报：自然科学版，2008，10(6)：18－111.

[9]　周宁，苗放，周丽.在 DAO 模式中实现数据库间差异消除及数据库操作的移植[J].计算机工程与科学，2006，28(10)：111－113.

[10]　欧阳宏基，葛萌，赵蕾.基于 JDBC 与设计模式的数据库连接池实现方法[J].计算机技术与发展，2011，21(1)：84－87.